Systèmes concurrents

Philippe Quéinnec

14 septembre 2012

Cinquième partie

L'interblocage

Plan

Propriétés temporelles

Pour *toutes* les exécutions possibles,
à *tout* moment d'une exécution.

- sûreté : rien de mauvais ne se produit
  l'exclusion mutuelle, les invariants d'un programme
- vivacité : quelque chose de bon finit par se produire
  l'équité, l'absence de famine, la terminaison d'une boucle
- (possibilité : pour *certaines* exécutions, . . .)

La vivacité

- Progression : si *des* processus déposent des requêtes de manière continue, *une* des requêtes finira par être satisfaite ;
- Vivacité faible : si un processus dépose sa requête de manière continue, elle finira par être satisfaite ;
- Vivacité forte : si un processus dépose une requête infiniment souvent, elle finira par être satisfaite ;
- Vivacité FIFO : si un processus dépose une requête, elle sera satisfaite avant tout autre requête (conflictuelle) déposée ultérieurement.
  Facile mais peu performante en centralisé, difficile en réparti.

(les processus sont supposés se comporter « correctement »)

La famine (privation)

Définition

Un processus est en famine lorsqu'il attend infiniment longtemps la satisfaction de sa requête (elle n'est jamais satisfaite).

Vivacité $\rightarrow$ absence de famine $\rightarrow$ absence d'interblocage

Plan

Allocation de ressources multiples

- Ressources banalisées, réutilisables, regroupées en classes
- Un processus demande un certain nombre de ressources dans chaque classe
- Demande bloquante, ressources allouées par un gérant de ressources
- Interface du gérant :
  - `demander` : (IdClasse → natural) → Set of IdRessource
  - `libérer` : Set of IdRessource → unit
- Le gérant :
  - rend la ressource réutilisable, lors d'une libération ;
  - libère les ressources détenues, à la terminaison d'un processus.

Exemples

Sémaphore

1 classe, R ressources, demande $= 1$

Philosophes

N classes de 1 ressource, P_i demande R_i et $R_{i \oplus 1}$

« Allocateur »

1 classe de R ressources, demande $\in [1..R]$

Lecteurs/rédacteurs

1 classe de N ressources, demande $= 1$ ou $= N$

Plan

L'interblocage

Allocation de ressources réutilisables

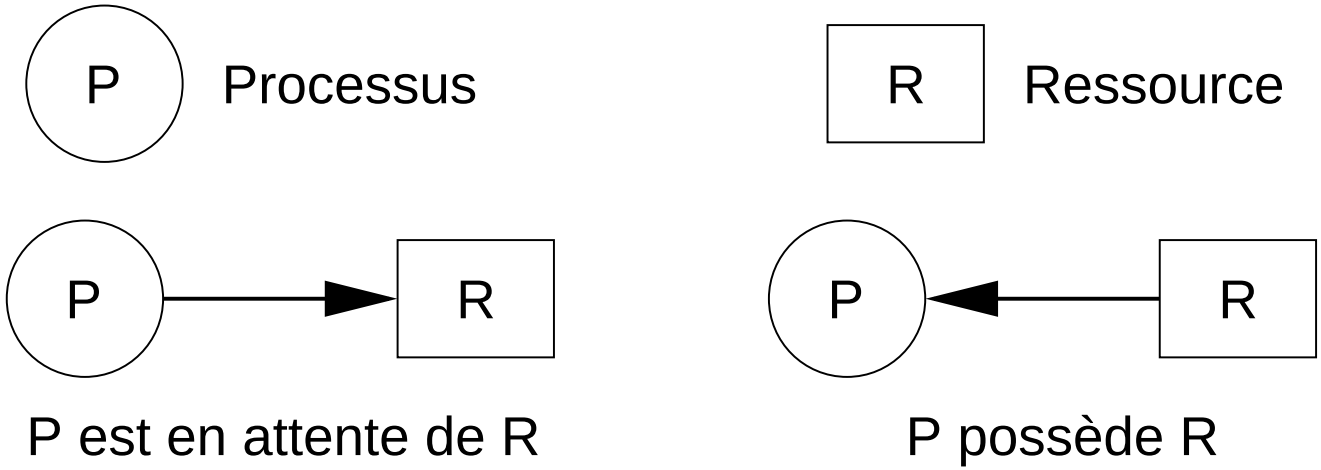
- non réquisitionnables
- non partageables
- en quantités entières et finies
- uniques ou multiples

Problème : P_1 demande A puis B,
P_2 demande B puis A
$\rightarrow$ risque d'interblocage.

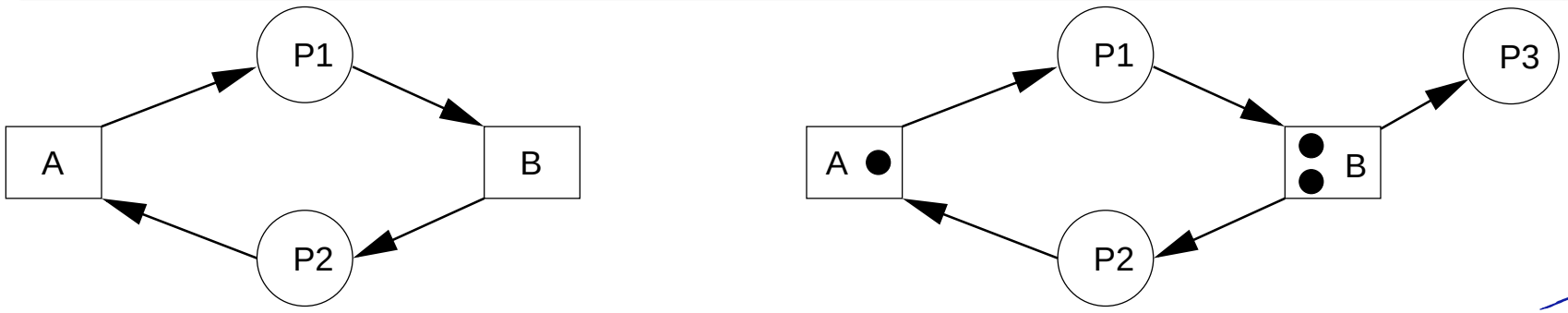
Définition

Un *ensemble* de processus est en interblocage si et seulement si tout processus de l'ensemble est en attente d'une ressource qui ne peut être libérée que par un autre processus de l'ensemble.

Graphe d'allocation



Interblocage ≡ cycle/knot dans le graphe d'allocation



Conditions nécessaires

L'interblocage est un état stable.

1. Les ressources sont utilisées en exclusion mutuelle
2. Les processus demandent plusieurs ressources en plusieurs fois :
   - pas de libération nécessaire avant une nouvelle demande
   - blocage tant que la requête ne peut pas être satisfaite
3. Ressources non réquisitionnables
4. Attente circulaire (cycle dans le graphe d'allocation)

Solutions

- Prévention : empêcher l'une des quatre conditions de survenir
- Détection + guérison : détecter l'interblocage, et l'éliminer

Prévention

Éviter l'accès exclusif

Ressource virtuelle : imprimante, fichier

Éviter la redemande bloquante

- Allocation globale : demander en une seule fois toutes les ressources nécessaires
  - connaissances a priori des ressources nécessaires
  - sur-allocation et risque de famine
- Acquisition non bloquante : le demandeur peut ajuster sa demande

Prévention

Éviter la non-réquisition

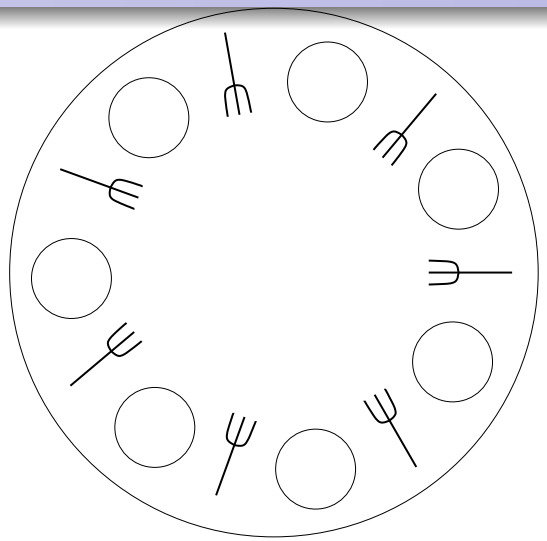
Avant de demander de nouvelles ressources, le processus est tenu de libérer toutes les ressources dont il dispose.
La libération de toutes les ressources n'est en réalité nécessaire que si la demande du processus est bloquante.

Éviter l'attente circulaire : classes ordonnées

- un ordre est défini sur les classes de ressources
- tout processus doit demander les ressources selon cet ordre

Solution utilisée pour éviter l'interblocage dû à des verrous multiples.

Exemple : Philosophes et spaghettis – Dijkstra



N philosophes sont autour d'une table.
Il y a une assiette par philosophe, et une fourchette entre chaque assiette.
Pour manger, un philosophe doit utiliser les deux fourchettes adjacentes à son assiette (et celles-là seulement).

Un philosophe peut être :

- penseur : il n'utilise pas de fourchettes ;
- mangeur : il utilise les deux fourchettes adjacentes ; aucun de ses voisins ne peut manger ;
- demandeur : il souhaite manger mais ne dispose pas des deux fourchettes.

Ce problème est analogue au problème de l'allocateur multi-classes multi-ressources.

Exemple : philosophes et interblocage

Risque d'interblocage

Chaque philosophe demande sa fourchette gauche et l'obtient. Puis quand tous ont leur fourchette gauche, chaque philosophe demande sa fourchette droite et se bloque. $\Rightarrow$ interblocage

Solutions

Allocation globale : chaque philosophe demande simultanément les deux fourchettes.

Non conservation : quand un philosophe essaye de prendre sa seconde fourchette et qu'elle est déjà prise, il relâche la première et se met en attente sur la seconde.

Classes ordonnées : imposer un ordre sur les fourchettes $\equiv$ tous les philosophes prennent d'abord la gauche puis la droite, sauf un qui prend d'abord droite puis gauche.

Esquive

Avant toute allocation, évaluation du risque (futur) d'interblocage.

L'algorithme du banquier

- Chaque processus annonce le nb max de ressources qu'il possédera.
- L'algorithme maintient le système dans un état fiable, i.e. tel qu'il existe toujours une possibilité d'éviter l'interblocage dans le pire des cas.
- En cas de danger détecté, le requête est mise en attente (comme s'il n'y avait pas les ressources).
- Un état est fiable si pour tout processus P_i, les ressources que P_i pourra demander peuvent être satisfaites avec les ressources actuellement disponibles + les ressources détenues par les P_j, $j < i$.

12 ressources, $P_0/P_1/P_2$ annoncent 10/4/9

	max	poss.	dem	
P_0	10	5		
P_1	4	2	+1	oui
				$(5 + \mathbf{4} + 2 \leq 12) \wedge (\mathbf{10} + 2 \leq 12) \wedge (\mathbf{9} \leq 12)$
P_2	9	2	+1	non
				$(\mathbf{10} + 2 + 3 > 12) \wedge (5 + 2 + \mathbf{9} > 12)$
				$\wedge((5 + \mathbf{4} + 3 \leq 12) \wedge (\mathbf{10} + 3 > 12) \wedge (5 + \mathbf{9} > 12))$

Détection

- Construire le graphe d'allocation
- Détecter l'existence d'un cycle (ressources uniques) ou d'un « knot » (ressources multiples équivalentes)

  knot = composante fortement connexe terminale
  = ensemble S de processus/ressources tels que
  $\forall s \in S$: successeur$(s) \in S$
  $\wedge\ s \in$ processus $\Rightarrow$ successeur$(s) \neq \emptyset$
  $\wedge\ s \in$ ressources $\Rightarrow$ card(succ(s)) = nb ress.
- Algorithme coûteux $\rightarrow$ périodiquement (et non à chaque allocation)

Guérison

Réquisition des ressources détenues par un (ou plusieurs) processus.

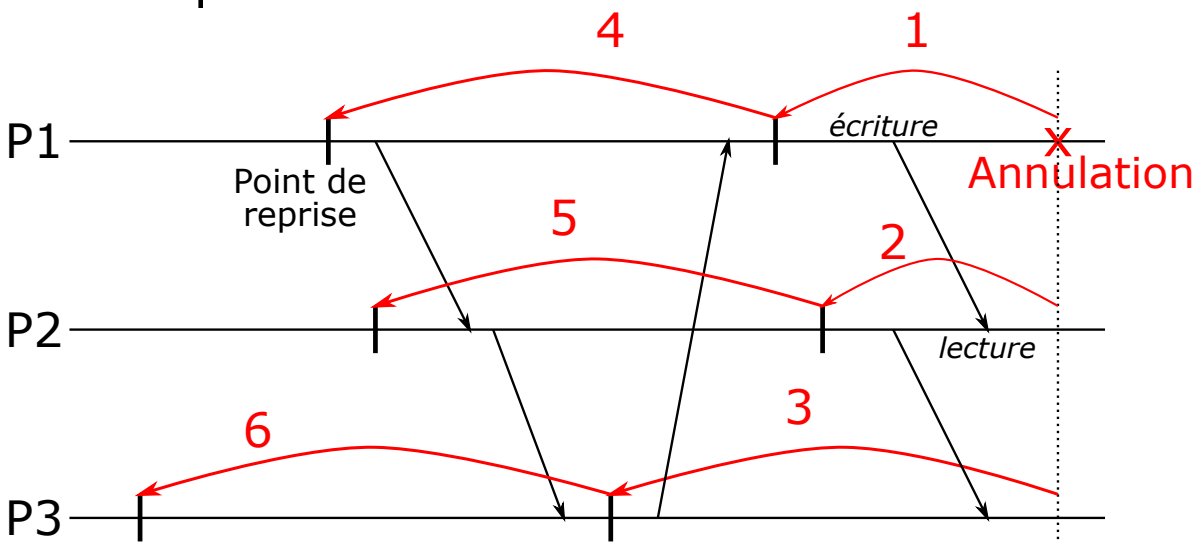
- Comment choisir le(s) processus victimes (le dernier qui a alloué, le plus gros/petit consommateur, notion d'importance...) ?
- Annulation du processus ou retour en arrière ?
- Solution coûteuse (détection + choix + travail perdu), pas toujours acceptable (systèmes interactifs, systèmes embarqués).
- Simplifie l'allocation.
- Points de reprise pour retour arrière.

Points de reprise

Définition

Sauvegarde d'états intermédiaires pour éviter de perdre tout le travail.

Utilisé pour les transactions en base de données
Effet domino : l'annulation d'une action entraîne l'annulation d'une deuxième action qui...



Plan

L'interblocage

- Usuellement : interblocage = inconvénient occasionnel
  - $\rightarrow$ laissé à la charge de l'utilisateur/du programmeur
  - utilisation de méthodes de prévention simples (p.e. classes ordonnées)
  - ou détection empirique et guérison par choix manuel des victimes
- Cas particuliers : systèmes ouverts (plus ou moins) contraints par le temps
  - systèmes interactifs, multiprocesseurs, systèmes embarqués
  - recherche de méthodes efficaces, prédictibles, ou automatiques
  - compromis/choix à réaliser entre la prévention (statique, coûteuse, restreint le parallélisme) et détection/guérison (moins prédictible, coûteuse quand les conflits sont fréquents).

Autres difficultés de synchronisation

- Accès non protégé à un état partagé (conflits d'écriture, visibilité d'états transitoires). Se manifestent souvent par des comportements non reproductibles (heisenbugs)
- Invalidation d'un invariant (souvent facile à détecter, mais coûteux et non correctible en général)
- Famine : difficilement prouvable, impossible à détecter (sauf stratégie régulière : FIFO)
- « Livelock » : processus bouclant sans jamais libérer ses ressources ou acquisition non bloquante avec retentatives qui échouent toujours.